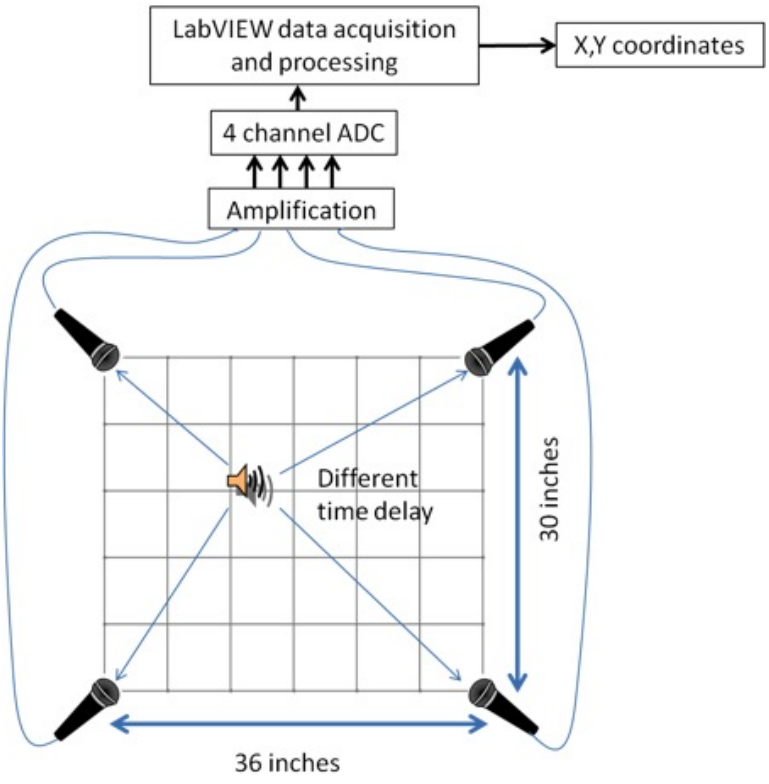
Introduction

Every object is capable to create sound, which is due to vibration of air. Using microphones, those difference in air pressure are transduced into electrical signal which can then be recorded and processed. Since the vibration of air decreases in amplitude relative with distance, and the vibrations travel with a fixed velocity through various materials. We had an idea of calculating those differences to localize on the sound source location. Detecting the location of a sound source has various applications. It is a basic function that could be implemented into many complex systems like in-door GPS, sound-nevigation robots, acoustic camera, entertainment user interface etc. Previous groups that worked on similar project either used high-cost equipment or received less accurate result. Our group wants to design a sound positioning system based on low cost equipment and a greater degree of precision. Our design consists of two parts: hardware setup and algorithm selection. Both of them will be discussed into details in the following sessions.

Hardware Setup
Hardware_Setup

Hardware Setup

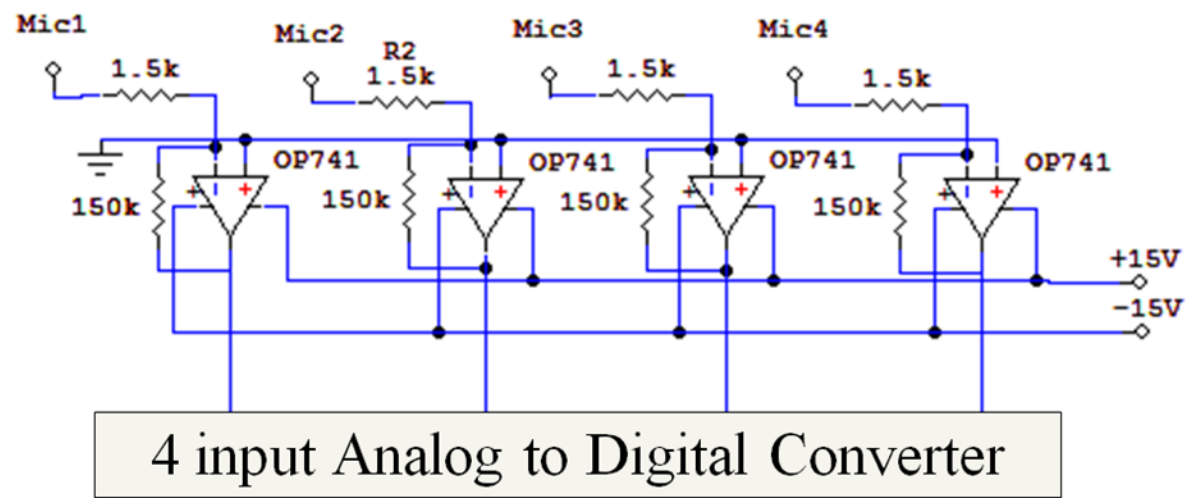
Our system is made up of 4 monochannel mics, a breadboard with 4 amplifiers (OP741), a 4 channel Analog to Digital converter (ADC, NI 9234) connected via a chasis (NI cDAQ-9174) to the computer through USB connectors. The overview of our system is shown in the figure below. Gathering of data from ADC is done through the LabVIEW program, and calculation of the x and y coordinates via the microphone delays is also done inside the program.



Microphones and Amplifiers

Since the microphones are monochannel, their sound transduced voltage data is output through a single 3.5mm audio jack. The amplifier circuit is done using OP741 with the circuit shown below, with a gain of -100X per channel. Gain is calculated through the following equation

$$Gain = -\frac{R_f}{R_1} = -\frac{150k}{1.5k} = -100$$

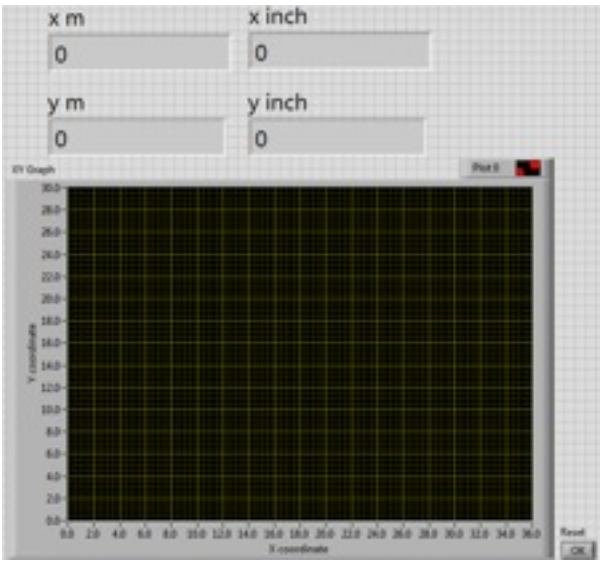


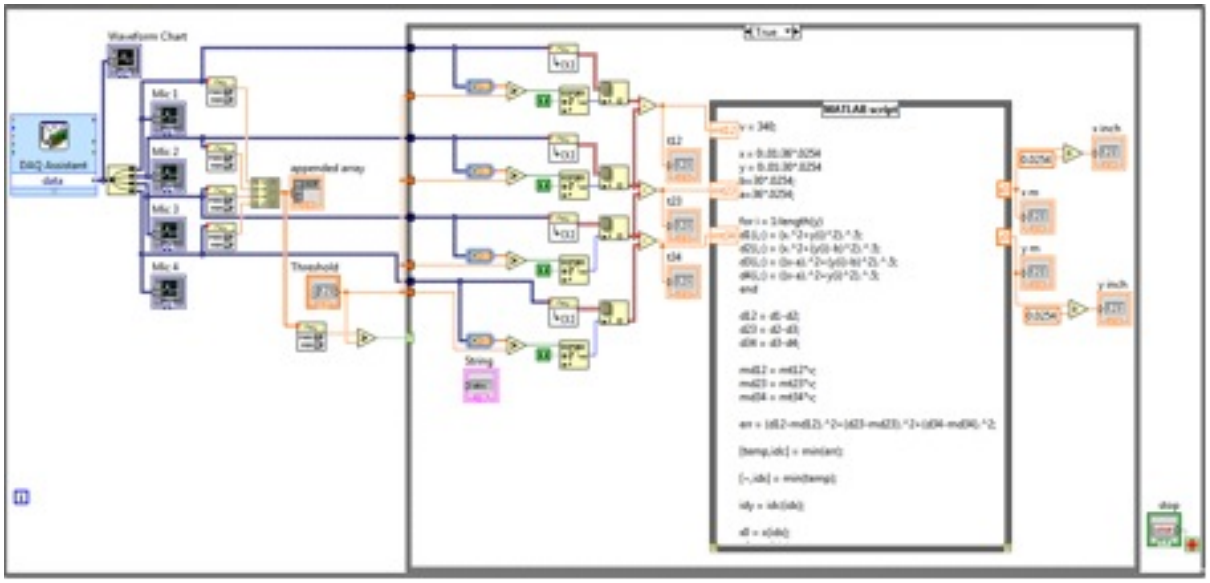
Analog to Digital Convertor

The Analog to Digital convertor (ADC) system is obtained from National Instrument. The input module is NI 9234 while the USB chasis is NI cDAQ-9174. The ADC is capable of obtaining up to 51.2kSamples/sec, which will allow us to process up to 39.0625μs of time difference accurately.

User Interface Setup

The labview interface is built using the system blocks shown below. Data is collected in chunks of 12.8k samples which correspond to 0.5seconds of acquisition per calculation. Using those data, we set a threshold value, which will trigger LabVIEW to output a the time at which the threshold is crossed. As each microphone picks up the sound at a different time, the time at which their voltage crosses the threshold is different. Using this delay, we are able calculate the x and y coordinate which will be output to a point in a graph.


x m
0
x inch
0
y m
0
y inch
0
XY Graph
Plot 0
Y coordinate
X coordinate
Reset
OK


Waveform Chart
DAQ Assistant
data
Mic 1
Mic 2
Mic 3
Mic 4
appended array
Threshold
String
True
MATLAB script
x inch
y inch
stop

Algorithm
Algorithm

Algorithm

We tried two methods that both can work with one of them having a better result. For both algorithms, we input the time delays between each microphone (microphone 1 through 4) and output the xy-coordinates.

Get Time Delays

First of all, to get the time delay information, we tried several methods to decide the time we get the pulse for each microphone. Method I is to catch the peak in certain time periods. In our Labview program, it checks every 0.5 second for at which time point the sound has the largest amplitude in the previous 0.5 second, and return that time point. Method II is to set an amplitude threshold, and return the first time point that is above the threshold. We set the threshold to be 0.5V to avoid the influence of noise and also catch the pulse we generate. In this method, we disable the threshold for 0.5 second after the program gets a time point to avoid the influence of imperfect pulse. By disabling the threshold, we ignore the sound that has amplitude above the threshold in this 0.5 second. After synchronizing the four microphones, we can get the time delays between each microphone.

Method I: Direct Equation Solving

We measured the time delay of received sound among four microphones, t_12, t_13, and t_14, where t_AB stands for the time delay of microphone A minus that of microphone B. Based on the physics fact that d=v*t and v is a constant (340m/s in the air), we know the relative distance difference from the point to four corners, called d_12, d_13, and d_14.

Assume the coordinates of the point we are solving to be (x,y), the length of the panel to be a, and the width of the panel to be b. Using some simple geometric fact, we can represent the distance from the point to each microphone with (x,y), a, and b. By equating the differences of the calculated distances to the corresponding measured differences, d_12, d_13, and d_14, we get three equations. We solve equation 1 and 2 in Matlab, and get two possible x's (called x1 and x2), and two possible y's (called y1 and y2). Similarly, by solving equation 2 and 3 in Matlab, we can get possible x's (called x3 and x4), and two possible y's (called y3 and y4). Theoretically, there's always a number that is both in the first set of x(x1 and x2) and the second set of x (x3 and x4), and that should be the x coordinate we want. The same holds for y coordinate.

In practice, due to the noise and errors, we can never get an exact same x or y in two sets. Therefore, we calculate the differences between x1 and x3, x1 and x4, x2 and x3, and x2 and x4, and find the pair with minimum difference. We then decide the average value of the pair to be the x coordinate. The same for y coordinate.

This method sometimes gives us results that far beyond our bounds and can be very unstable, so we try to develop a second method.

```
function [x,y] = coor(t12,t13,t14,a,b)

d1=340*t12;d2=340*t13;d3=340*t14; % distance differences

% The result from sovling physics equations using Matlab symbolic toolbox

x1 =[

 (a^2 - 2*d1*d2 + d1^2 + d2^2 - (2*d1*(a^2*d1^3 + b^2*d1^3 - b^2*d2^3 ...
     + a^2*b^2*d2 - 2*a^2*d1^2*d2 + 3*b^2*d1*d2^2 - 3*b^2*d1^2*d2 + ...
     a*b*(a^4*b^2 - a^4*d1^2 + a^2*b^4 - 2*a^2*b^2*d1^2 + 2*a^2*b^2*d1*d2...
     - 2*a^2*b^2*d2^2 + a^2*d1^4 - 2*a^2*d1^3*d2 + 2*a^2*d1^2*d2^2 -...
     b^4*d1^2 + 2*b^4*d1*d2 - b^4*d2^2 + b^2*d1^4 - 2*b^2*d1^3*d2 + ...
     2*b^2*d1^2*d2^2 - 2*b^2*d1*d2^3 + b^2*d2^4 - d1^4*d2^2 + 2*d1^3*d2^3-...
     d1^2*d2^4)^(1/2)))/(- 2*a^2*b^2 + 2*a^2*d1^2 + 2*b^2*d1^2 - ...
     4*b^2*d1*d2 + 2*b^2*d2^2) + (2*d2*(a^2*d1^3 + b^2*d1^3 - b^2*d2^3 +...
     a^2*b^2*d2 - 2*a^2*d1^2*d2 + 3*b^2*d1*d2^2 - 3*b^2*d1^2*d2 + ...
     a*b*(a^4*b^2 - a^4*d1^2 + a^2*b^4 - 2*a^2*b^2*d1^2 + ...
     2*a^2*b^2*d1*d2 - 2*a^2*b^2*d2^2 + a^2*d1^4 - 2*a^2*d1^3*d2 + ...
     2*a^2*d1^2*d2^2 - b^4*d1^2 + 2*b^4*d1*d2 - b^4*d2^2 + b^2*d1^4 - ...
     2*b^2*d1^3*d2 + 2*b^2*d1^2*d2^2 - 2*b^2*d1*d2^3 + b^2*d2^4 - ...
     d1^4*d2^2 + 2*d1^3*d2^3 - d1^2*d2^4)^(1/2)))/(- 2*a^2*b^2 + ...
     2*a^2*d1^2 + 2*b^2*d1^2 - 4*b^2*d1*d2 + 2*b^2*d2^2))/(2*a)
 (a^2 - 2*d1*d2 + d1^2 + d2^2 - (2*d1*(a^2*d1^3 + b^2*d1^3 - b^2*d2^3 + ...
 a^2*b^2*d2 - 2*a^2*d1^2*d2 + 3*b^2*d1*d2^2 - 3*b^2*d1^2*d2 - ...
 a*b*(a^4*b^2 - a^4*d1^2 + a^2*b^4 - 2*a^2*b^2*d1^2 + 2*a^2*b^2*d1*d2 -...
 2*a^2*b^2*d2^2 + a^2*d1^4 - 2*a^2*d1^3*d2 + 2*a^2*d1^2*d2^2 - b^4*d1^2 +...
 2*b^4*d1*d2 - b^4*d2^2 + b^2*d1^4 - 2*b^2*d1^3*d2 + 2*b^2*d1^2*d2^2 - ...
 2*b^2*d1*d2^3 + b^2*d2^4 - d1^4*d2^2 + 2*d1^3*d2^3 - d1^2*d2^4)^(1/2)))/...
 (- 2*a^2*b^2 + 2*a^2*d1^2 + 2*b^2*d1^2 - 4*b^2*d1*d2 + 2*b^2*d2^2) + ...
 (2*d2*(a^2*d1^3 + b^2*d1^3 - b^2*d2^3 + a^2*b^2*d2 - 2*a^2*d1^2*d2 + ...
 3*b^2*d1*d2^2 - 3*b^2*d1^2*d2 - a*b*(a^4*b^2 - a^4*d1^2 + a^2*b^4 - ...
 2*a^2*b^2*d1*d2 - 2*a^2*b^2*d2^2 + a^2*d1^4 - 2*a^2*d1^3*d2 + ...
 2*a^2*d1^2*d2^2 - b^4*d1^2 + 2*b^4*d1*d2 - b^4*d2^2 + b^2*d1^4 - ...
 2*b^2*d1^3*d2 + 2*b^2*d1^2*d2^2 - 2*b^2*d1*d2^3 + b^2*d2^4 - ...
 d1^4*d2^2 + 2*d1^3*d2^3 - d1^2*d2^4)^(1/2)))/(- 2*a^2*b^2 + ...
 2*a^2*d1^2 + 2*b^2*d1^2 - 4*b^2*d1*d2 + 2*b^2*d2^2))/(2*a)];


y1 =[

 (2*d1*d2 + b^2 - d1^2 + (2*d1*(a^2*d1^3 + b^2*d1^3 - b^2*d2^3 + ...
 a^2*b^2*d2 - 2*a^2*d1^2*d2 + 3*b^2*d1*d2^2 - 3*b^2*d1^2*d2 + ...
 a*b*(a^4*b^2 - a^4*d1^2 + a^2*b^4 - 2*a^2*b^2*d1^2 + 2*a^2*b^2*d1*d2 - ...
 2*a^2*b^2*d2^2 + a^2*d1^4 - 2*a^2*d1^3*d2 + 2*a^2*d1^2*d2^2 - b^4*d1^2 + ...
 2*b^4*d1*d2 - b^4*d2^2 + b^2*d1^4 - 2*b^2*d1^3*d2 + 2*b^2*d1^2*d2^2 - ...
 2*b^2*d1*d2^3 + b^2*d2^4 - d1^4*d2^2 + 2*d1^3*d2^3 - d1^2*d2^4)^(1/2)))/...
 (- 2*a^2*b^2 + 2*a^2*d1^2 + 2*b^2*d1^2 - 4*b^2*d1*d2 + 2*b^2*d2^2))/(2*b);
 (2*d1*d2 + b^2 - d1^2 + (2*d1*(a^2*d1^3 + b^2*d1^3 - b^2*d2^3 + ...
 a^2*b^2*d2 - 2*a^2*d1^2*d2 + 3*b^2*d1*d2^2 - 3*b^2*d1^2*d2 - ...
 a*b*(a^4*b^2 - a^4*d1^2 + a^2*b^4 - 2*a^2*b^2*d1^2 + 2*a^2*b^2*d1*d2 - ...
 2*a^2*b^2*d2^2 + a^2*d1^4 - 2*a^2*d1^3*d2 + 2*a^2*d1^2*d2^2 - b^4*d1^2 + ...
 2*b^4*d1*d2 - b^4*d2^2 + b^2*d1^4 - 2*b^2*d1^3*d2 + 2*b^2*d1^2*d2^2 - ...
 2*b^2*d1*d2^3 + b^2*d2^4 - d1^4*d2^2 + 2*d1^3*d2^3 - d1^2*d2^4)^(1/2)))/...
 (- 2*a^2*b^2 + 2*a^2*d1^2 + 2*b^2*d1^2 - 4*b^2*d1*d2 + 2*b^2*d2^2))/(2*b)];
```

```
x2 =[

 (a*(d2*d3^3 + d2^3*d3 + 2*b^2*d3^2 - 2*d2^2*d3^2 - b^2*d2*d3 - (2*b^2*...
 d3^2*(b^2 - d2^2 + 2*d2*d3 - d3^2))/(2*b^2 - 2*d2^2 + 4*d2*d3 - 2*d3^2)) +...
 b*d3*(a^4*b^2 - a^4*d2^2 + 2*a^4*d2*d3 - a^4*d3^2 + a^2*b^4 - 2*a^2*b^2*...
 d2^2 + 2*a^2*b^2*d2*d3 - 2*a^2*b^2*d3^2 + a^2*d2^4 - 2*a^2*d2^3*d3 + 2*...
 a^2*d2^2*d3^2 - 2*a^2*d2*d3^3 + a^2*d3^4 - b^4*d3^2 + 2*b^2*d2^2*d3^2 - ...
 2*b^2*d2*d3^3 + b^2*d3^4 - d2^4*d3^2 + 2*d2^3*d3^3 - d2^2*d3^4)^(1/2))/...
 (d3^2*(2*a^2 + 2*b^2) - 2*a^2*b^2 + 2*a^2*d2^2 - 4*a^2*d2*d3) + (a*...
 (b^2 - d2^2 + 2*d2*d3 - d3^2))/(2*b^2 - 2*d2^2 + 4*d2*d3 - 2*d3^2);
 (a*(d2*d3^3 + d2^3*d3 + 2*b^2*d3^2 - 2*d2^2*d3^2 - b^2*d2*d3 - (2*b^2*...
 d3^2*(b^2 - d2^2 + 2*d2*d3 - d3^2))/(2*b^2 - 2*d2^2 + 4*d2*d3 - 2*d3^2))-...
 b*d3*(a^4*b^2 - a^4*d2^2 + 2*a^4*d2*d3 - a^4*d3^2 + a^2*b^4 - 2*a^2*b^2*...
 d2^2 + 2*a^2*b^2*d2*d3 - 2*a^2*b^2*d3^2 + a^2*d2^4 - 2*a^2*d2^3*d3 +...
 2*a^2*d2^2*d3^2 - 2*a^2*d2*d3^3 + a^2*d3^4 - b^4*d3^2 + 2*b^2*d2^2*d3^2 -...
 2*b^2*d2*d3^3 + b^2*d3^4 - d2^4*d3^2 + 2*d2^3*d3^3 - d2^2*d3^4)^(1/2))/...
 (d3^2*(2*a^2 + 2*b^2) - 2*a^2*b^2 + 2*a^2*d2^2 - 4*a^2*d2*d3) + (a*...
 (b^2 - d2^2 + 2*d2*d3 - d3^2))/(2*b^2 - 2*d2^2 + 4*d2*d3 - 2*d3^2)];

 y2 =[

 (b^2 - 2*d2*d3 + d2^2 + d3^2 + (2*d2*(a^2*d3^3 - a^2*d2^3 + b^2*d3^3 + ...
 a^2*b^2*d2 - 3*a^2*d2*d3^2 + 3*a^2*d2^2*d3 - 2*b^2*d2*d3^2 + a*b*...
 (a^4*b^2 - a^4*d2^2 + 2*a^4*d2*d3 - a^4*d3^2 + a^2*b^4 - 2*a^2*b^2*d2^2+...
 2*a^2*b^2*d2*d3 - 2*a^2*b^2*d3^2 + a^2*d2^4 - 2*a^2*d2^3*d3 + 2*a^2*...
 d2^2*d3^2 - 2*a^2*d2*d3^3 + a^2*d3^4 - b^4*d3^2 + 2*b^2*d2^2*d3^2 - 2*...
 b^2*d2*d3^3 + b^2*d3^4 - d2^4*d3^2 + 2*d2^3*d3^3 - d2^2*d3^4)^(1/2)))/...
 (- 2*a^2*b^2 + 2*a^2*d2^2 - 4*a^2*d2*d3 + 2*a^2*d3^2 + 2*b^2*d3^2) - ...
 (2*d3*(a^2*d3^3 - a^2*d2^3 + b^2*d3^3 + a^2*b^2*d2 - 3*a^2*d2*d3^2 + ...
 3*a^2*d2^2*d3 - 2*b^2*d2*d3^2 + a*b*(a^4*b^2 - a^4*d2^2 + 2*a^4*d2*d3 - ...
 a^4*d3^2 + a^2*b^4 - 2*a^2*b^2*d2^2 + 2*a^2*b^2*d2*d3 - 2*a^2*b^2*d3^2 + ...
 a^2*d2^4 - 2*a^2*d2^3*d3 + 2*a^2*d2^2*d3^2 - 2*a^2*d2*d3^3 + a^2*d3^4 - ...
 b^4*d3^2 + 2*b^2*d2^2*d3^2 - 2*b^2*d2*d3^3 + b^2*d3^4 - d2^4*d3^2 + 2*...
 d2^3*d3^3 - d2^2*d3^4)^(1/2)))/(- 2*a^2*b^2 + 2*a^2*d2^2 - 4*a^2*d2*d3 + ...
 2*a^2*d3^2 + 2*b^2*d3^2))/(2*b);
 (b^2 - 2*d2*d3 + d2^2 + d3^2 - (2*d2*(a^2*d2^3 - a^2*d3^3 - b^2*d3^3 - ...
 a^2*b^2*d2 + 3*a^2*d2*d3^2 - 3*a^2*d2^2*d3 + 2*b^2*d2*d3^2 + a*b*(a^4*...
 b^2 - a^4*d2^2 + 2*a^4*d2*d3 - a^4*d3^2 + a^2*b^4 - 2*a^2*b^2*d2^2 + 2*...
 a^2*b^2*d2*d3 - 2*a^2*b^2*d3^2 + a^2*d2^4 - 2*a^2*d2^3*d3 + 2*a^2*d2^2*...
 d3^2 - 2*a^2*d2*d3^3 + a^2*d3^4 - b^4*d3^2 + 2*b^2*d2^2*d3^2 - 2*b^2*d2*...
 d3^3 + b^2*d3^4 - d2^4*d3^2 + 2*d2^3*d3^3 - d2^2*d3^4)^(1/2)))/(- 2*a^2*...
 b^2 + 2*a^2*d2^2 - 4*a^2*d2*d3 + 2*a^2*d3^2 + 2*b^2*d3^2) + (2*d3*(a^2*...
 d2^3 - a^2*d3^3 - b^2*d3^3 - a^2*b^2*d2 + 3*a^2*d2*d3^2 - 3*a^2*d2^2*d3 + ...
 2*b^2*d2*d3^2 + a*b*(a^4*b^2 - a^4*d2^2 + 2*a^4*d2*d3 - a^4*d3^2 + a^2*...
 b^4 - 2*a^2*b^2*d2^2 + 2*a^2*b^2*d2*d3 - 2*a^2*b^2*d3^2 + a^2*d2^4 - 2*...
 a^2*d2^3*d3 + 2*a^2*d2^2*d3^2 - 2*a^2*d2*d3^3 + a^2*d3^4 - b^4*d3^2 + 2*...
 b^2*d2^2*d3^2 - 2*b^2*d2*d3^3 + b^2*d3^4 - d2^4*d3^2 + 2*d2^3*d3^3 - ...
 d2^2*d3^4)^(1/2)))/(- 2*a^2*b^2 + 2*a^2*d2^2 - 4*a^2*d2*d3 + 2*a^2*...
 d3^2 + 2*b^2*d3^2))/(2*b)];

% method for choose the one possible correct solution out of four

xd=[x1(1)-x2(1);x1(1)-x2(2);x1(2)-x2(1);x1(2)-x2(2)];
yd=[y1(1)-y2(1);y1(1)-y2(2);y1(2)-y2(1);y1(2)-y2(2)];
[~,xid]=min(abs(xd));
[~,yid]=min(abs(yd));
```

```
if xid==1
    x=(x1(1)+x2(1))/2;
elseif xid==2
    x=(x1(1)+x2(2))/2;
elseif xid==3
    x=(x1(2)+x2(1))/2;
elseif xid==4
    x=(x1(2)+x2(2))/2;
end

if yid==1
    y=(y1(1)+y2(1))/2;
elseif yid==2
    y=(y1(1)+y2(2))/2;
elseif yid==3
    y=(y1(2)+y2(1))/2;
elseif yid==4
    y=(y1(2)+y2(2))/2;
end

end
```

Published with MATLAB® 7.12

Method II: Match Filter

Setting the resolution to be 1 cm, we divide the board to 1cm*1cm grid. For each point (m, n) on the grid, calculate the distance from this point to each microphone using geometric relations, and get the distance differences d_12, d_23, and d_34, where d_AB stands for the distance to microphone A minus distance to microphone B. Note that we use d_23 and d_34 instead of d_13 and d_14 as in the first method since we notice that it may introduce more error if we rely on one microphone (microphone 1) too much. Based on the physics fact that t=d/v and v is a constant (340m/s in the air), we get the theoretical time delays t_12, t_23, and t_34 for point (m, n).

Assume the corresponding measured time delay to be m_12, m_23, and m_34. For each point on the grid, we calculate the root-square-mean error between the measured time delays and the theoretical time delays:

$$\sqrt{\frac{(t_{12}-m_{12})^2+(t_{23}-m_{23})^2+(t_{34}-m_{34})^2}{3}}$$

We then find the point (x, y) in the grid that has the least error and return it as the xy coordinate.

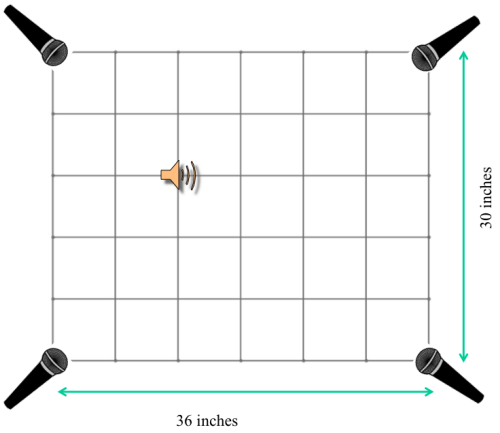
```
function [x0 y0] = findCo(a,b,res,mt12,mt23,mt34)
v = 340; % sound speed in the air

x = 0:res:a; % set system prefixed resolution
y = 0:res:b;

for i = 1:length(y) % distance from one point to four microphones
    d1(i,:) = (x.^2+y(i)^2).^.5;
    d2(i,:) = (x.^2+(y(i)-b)^2).^.5;
    d3(i,:) = ((x-a).^2+(y(i)-b)^2).^.5;
    d4(i,:) = ((x-a).^2+y(i)^2).^.5;
end


d12 = d1-d2; % the difference among distances
d23 = d2-d3;
d34 = d3-d4;

md12 = mt12*v; % measured distance differences
md23 = mt23*v;
md34 = mt34*v;

% generate the error matix to compare the three diferences above
err = (d12-md12).^2+(d23-md23).^2+(d34-md34).^2;

% choose the point that returns the smallest error
[temp,idc] = min(err);

[~,idx] = min(temp);

idy = idc(idx);

x0 = x(idx);
y0 = y(idy);

end
```

Published with MATLAB® 7.12

Testing
In this module, we show the testing results and compare those of different algorithm.

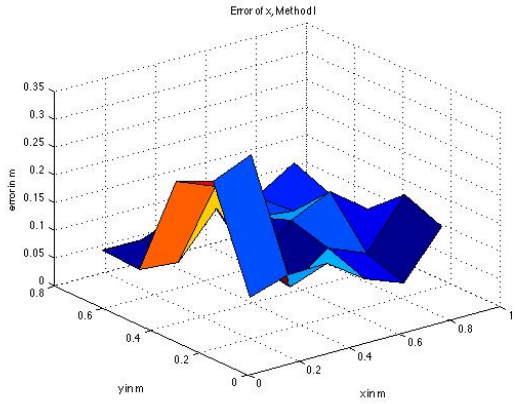
Testing

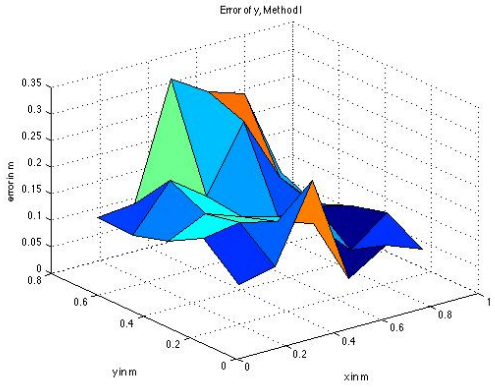
In order to make direct comparison between two algorithm and two detecting methods that we have discussed in the previous module, we conducted several tests. During each test, we used a 36in*30in board and drew grids every 6 inches. At each intersection point of grids, we generated a sound pulse. On the purpose of producing ideal pulse waves, we knocked two small brass blocks. Six trails were conducted at each point. We record both sound sources' coordinates and the ones our system measured. After computing the root-mean-square error of six trails at each point, we plot error maps for both x and y coordinates of the testing board.



Testing Results

Method I, Direct Equation Solving Method (Threshold Detecting)

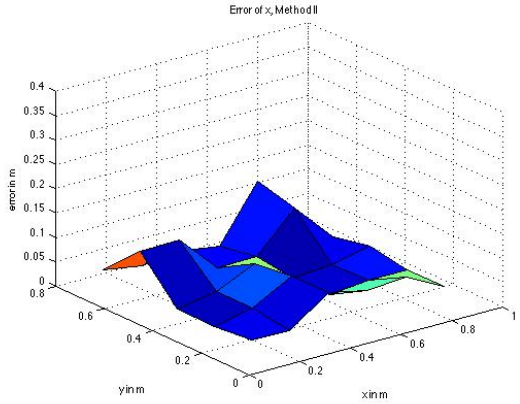


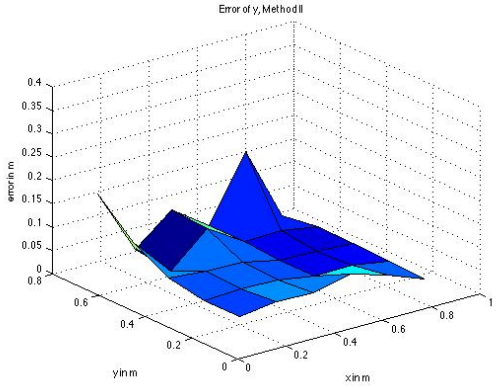


As shown in the error map above, this method returns coordinates with large errors. It has root-square-mean error 0.110m for x coordinates and 0.133m for y coordinates. Another problem for this method is that the results from six trails at one point have large variance. The performance of systems is unstable when using *Direct Equation Solving Method*.

The later analysis shows us that the errors mainly came two aspects. First, we used Matlab build in symbolic toolbox to solve the equation groups, which returns four possible solutions. Despite the fact that the true coordinate lies within those four solutions, there seems no good method that always chooses the right one. Second, the direct physics calculations require high accuracy from hardware. Directly plugging the measured data into equation, the errors are introduced and added up from all the hardware. Therefore, *Method I*, *Direct Equation Solving Method* could not fulfill our requirement of high resolution.

Method II, Match Filter (Threshold Detecting)

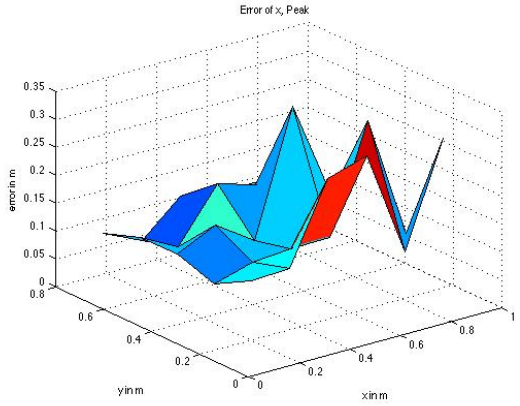


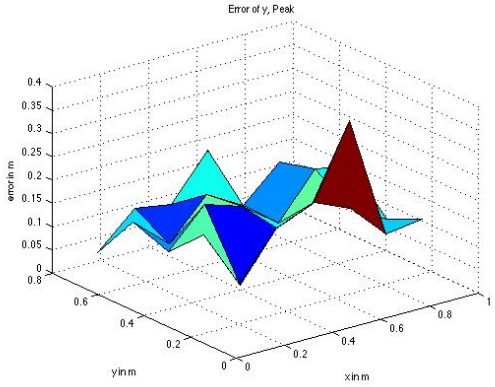


As shown in the error map above, this method returns coordinates with small errors. It has root-square-mean error 0.035m for x coordinates and 0.043m for y coordinates. Considering the prefixed resolution of our system is 0.01m, our system's provides good accuracy under *Method II, Match Filter (Threshold Detecting).*

In this algorithm, we avoided problems that causes large errors in method I. The calculations are numerical and only one result will be returned. The comparison algorithm also avoid calculating result by direct measured data and therefore control the errors coming from hardware. The improvement in performance matches our expectation.

Method II, Match Filter (Peak Detecting)





As shown in the error map above, this method returns coordinates with large errors. It has root-square-mean error 0.138m for x coordinates and 0.136m for y coordinates. Different form Method I, the 6 trails at one point returned relative consistent result. Because of the large errors it returned, this method can not fulfill our resolusion requirements.

Even though using the same match filter algorithm, peak detecting method seems to have a fixed error. Our later analysis attributes this error to the resonance and wave form distortion during transimitting in the air.

Conclusion

Conclusion

Form the testing data, the minimum x,y coordinate error our system would return is 0.035m and 0.043m respectively. This accuracy level satisfies our resolution requirement and the good precision degree could be contributed into both hardware implement and algorithm choice.

On the hardware level, our microphone, amplifier, and 4 channel simultaneous analog-to-digital function well. They effectively catch the sound pulse signal from the board and suppress the other noise from external environment. The Labview interface provides user flexible way to control hardware and choose from different algorithms. By adjusting threshold voltage, sampling rate, sample length, we have control on system's sensitivity, noise resistivity and computational speed. The optimal combination we found through several tests lead us to the final accurate result.

On the algorithm level, we have compared of two different method, *Direct Equation Solving,* and *Match Filter.* Both testing result and theoretical analysis have shown us that the latter offers us better result. The Match Filter method avoids the two major disadvantages of Direct Equation Solving. First, it does not need to select solution from a solution set and second, it reduces hardware error by making comparison and choosing the one with least returning error. Other advantages of Match Filter method will be discussed into more details in Future Work session. We also compared two detecting method: *Threshold Detecting* and *Peak Detecting* . The testing result shows that Threshold Detecting provided us much better results. Our analysis shows that the inaccuracy of Peak Detecting may be caused by resonance and waveform distortion during transmission.

In a nutshell, our sound positioning system with *Match Filter* algorithm and *Threshold Detecting* method returns us the most accurate xy coordinates. The high accuracy (around 4cm error out of 1cm resolution) gives us enough confidence to implement our sound positioning system into further various applications. However, there also exist several limits for our system.

Both those limits and applications will be discussed in the next session, Future Work.

Future Work

Future work

Improvement:

To get a higher accuracy, we can use more microphones. Since we use the match filter method, more microphones means more information and we should get a better result. Also, we can use the match filter heat map to decide the coordinates. In our current algorithm, we calculate the error between the measured time delay and the theoretical time delay. Instead, in the improved algorithm, we get the whole sound signal in a time period. For each point in the match filter heat map, we delay the sound signal we get in our measurement according to the theoretical time delay for each microphone, and calculate the inner product of the delayed signal and the measured signal for the microphone. For each point, we add up the inner products for the four microphones. We then find the point that generates the maximum sum of inner product and decide it as the coordinate. We come up with this idea very late so we do not have time to set it up and do some testing, so we save it as a possible improvement.

By using extra microphones we can easily add a third dimension to the equation and be able to detect objects in a 3D space. If we put microphones not in a 2D plane but in 3D space, and also divide the space into 1cm*1cm*1cm grid, we can implement a 3D system with our current algorithm.

We also want to detect sound that is not a pulse. In our current algorithm, our results still rely on the quality, such as sharpness, of the pulse. It may be related to the method we use to get the time delay. The threshold detecting needs a sudden rise on the amplitude to get a good result. To detect sound other than a pulse, we may need to make some adjustment to the way we decide the time delay.

Another constraint in our system is that the sound must be mainly transmitted through air since we use the fact that v=340m/s in our algorithm. We want to use the system on all kind of materials. In our

testing, we find out that when we step on the floor instead of clapping hands, the results are not accurate. Since the sound speed in different materials can be very different, we need to find a way to avoid using the sound speed in our algorithm or cancel the effect of the sound speed difference.

Possible Applications:

Anyone remember Billie Jean's music video by Michael Jackson, the sidewalk that lit up would be a very simple application to do even with our current setup.

Another possible application could be lighting in a house, By setting up microphones inside the house we could detect which rooms are in use and have the lights in that room turn on and off as people come and go. If we develop the system in 3D space, we can set different areas to control all the lights in a house. For example, the area around door is for the desk lamp, the area around table is for the crystal lamp, etc.

When we achieve better accuracy and get a more sensitive system, we can even develop a keyboard. By attaching several microphones to a piece of paper with keyboard pattern, we get a foldable and portable and cheap keyboard. We can simply put it on a hard surface and use a stylus to type.

Also, we can make an instrument for entertainment based on our system. Designating a note for each area on the panel, and translating the xy coordinate into different pitch, we can easily make an instrument. Actually, we have made a simple instrument when we tested our system. We translated the xy coordinate into different pitch, and input the pitch into Matlab to generate the sound.